Week 13 : Approaching open ended DS problems

pending tasks	
type	

Handling missing data

Using Numpy

- Reduce operation here means, reducing the 1D vector array to a scalar number (eg. sum). np.nan returns nan while using reduction and None returns a Runtime error.
- A boolean mask can be used along with fancy indexing to consider only the valid data. Below shown is an example for the same.

```
x = np.array([1, 2, 3, '--', 5])
x_b = np.array([True, True, True, False, True])
x[x_b].mean()
>>> 2.75
# using numpy submodule
m_x = np.ma.masked_array(x, mask = [0, 0, 0, 1, 0])
m_x.mean()
>>> 2.75
```

- Masking is not supported in pandas.

Missing data with pandas

- While reading the data, pandas replaces NA and n/a blank values as NaN.
- Object types take longer for processing.
- df.isnull().sum() returns the number of missing values in each column.
- A list of values can be passed while reading the dataset into pandas dataframe using na_values parameter. The elements of the list, if found in the dataframe, are converted to NaN.

```
missing_values = ["NA", "n/a", "na"]
df = pd.read_csv("rooms.csv",
                 na_values = missing_values)
```

- The masked out values are not used while calculating the statistics such as mean.

Filling missing values

- fillna() can be used to replace NaN with another value.

```
df.Occupied.fillna("N", inplace=True)
# using the previous row value
df.Department.fillna(method="pad", inplace=True)
# using the value from the next row
df.Dept2.fillna(method="bfill", inplace=True)
# using a summary statistic
df.Num_Students.fillna(df.Num_Students.median(), inplace=True)
# filling missing values within a range of numbers
df.Room_Number.interpolate(inplace=True)
```

- Interpolate can be used to fill up higher order polynomials too.

Open ended descriptive statistics

Looking at the approach to open ended tasks in descriptive statistics. The ameo_2015 data set found at http://research.aspiringminds.com/resources/#datasets was taken.

- The first few things to check would be a sample of the dataset (df.head()), the shape, number of null values at column level and dataset level and the data types of each column.

1. Checking gender bias - approach

- Check the unique values in the column gender.
- Try a plot say violin plot for gender vs salary.
- Check some statistics. Based on GPA in 10, 12 and college, and personality traits find the mean (or median) for each gender(use groupby).
- Mean for Salary vs Gender seemed to suggest that there's a gender bias. But the violin plot is not very different . Thus, checking the mean of higher and lower pay brackets individually.
- The threshold for a higher salary bracket can be the values one standard deviation away, to the right of the mean.
- It can be seen that the number of working women in general is less and is significantly lesser in the higher pay prabket. Thus, leading to the difference in the overall mean.

```
th = df.Salary.mean()+df.Salary.std()
df['HighIncome'] = (df.Salary > th)
df[['Salary', 'HighIncome', 'Gender']].groupby(['HighIncome', 'Gender']).mean()
```

		Salary
HighIncome	Gender	
False	f	271499.454744
	m	272598.433606
True	f	832250.000000
	m	785344.827586

```
df[['Salary', 'HighIncome', 'Gender']].groupby(['HighIncome', 'Gender']).count()
```

		Salary
HighIncome	Gender	
False	f	917
	m	2809
True	f	40
	m	232

```
print('Low income female percentage', 917/(2809+917)*100)
>>> Low income female percentage 24.610842726784757
print('High income female percentage', 40/(232+40)*100)
>>> High income female percentage 14.705882352941178
```

2. Explaining the difference

- Comparing the salary brackets with CollegeTier shows that the fraction of employees from tier 2 has reduced in the higher income bracket.
- Comparing CollegeTier and Gender shows that the female percentage in college tier one is just 17 in tier 1. This shows that there are a lesser number of women in tier1 colleges where there is a higher possibility of falling under the higher income bracket.
- So CollegeTier seems to be one of the deciding factors of the higher income bracket.

Agriculture example part1

- The data is at year-season-district level and has area and production for a given crop.
- The null values in the dataset, '=' in 'Production' column is handled by specifying it in na_values while reading the file and dropping the rows with null values using dropna().

Visualizing - how different states have fared across the years

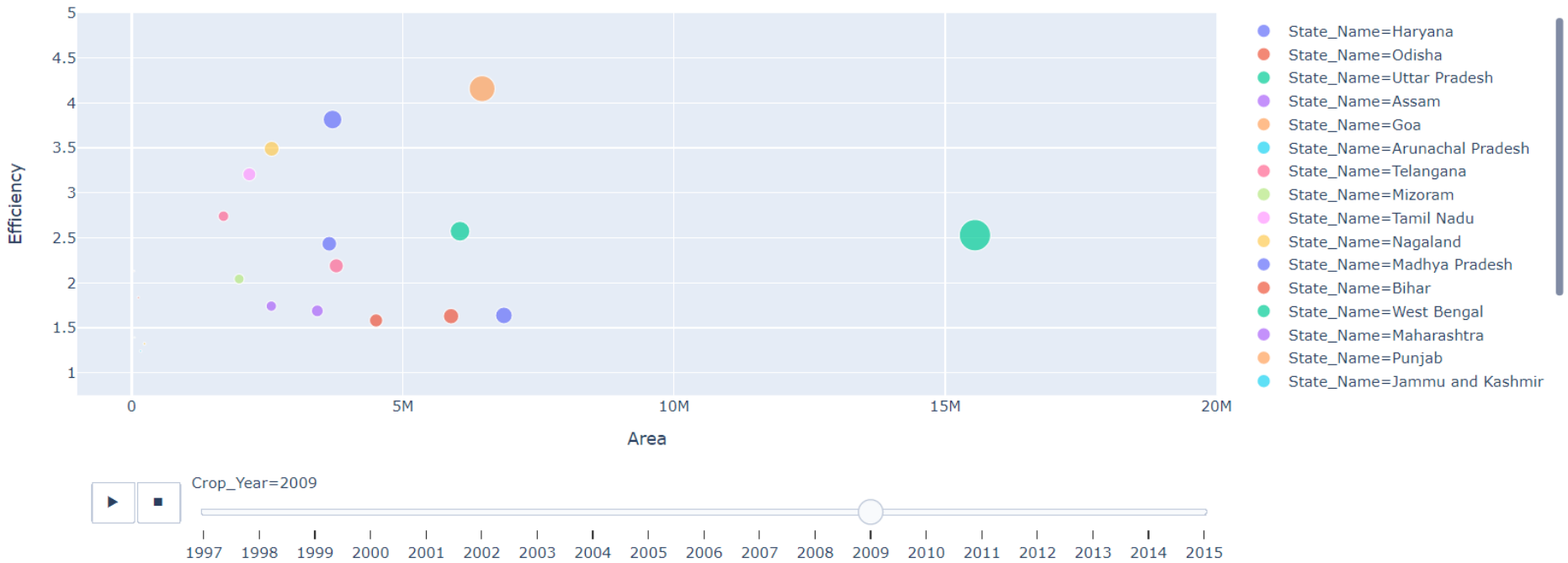
- The kde and box plot of production and area show that these values are left skewed.
- The visualization requires the area and production of crops in a given state on a given year.

- Line plots cannot be used as there are too many states, thus, dynamic plotting using plotly_express is used. The plot has scatter plots of area vs production for each year, as a sliding bar is used to slide through the years.

Agriculture Example part 2

- The data is filtered for few crops, this is to make the analysis easier and to remove outliers such as coconut whose contribution is significantly high to production since it is taken as weight produced.
- A column called efficiency is defined as production per unit area and plotted in the y axis, the production is used to depict the size of the bubbles.
- The range in X and Y axis is adjusted to keep the animation background fixed.

```
px.scatter(df_, x="Area", y="Efficiency", size="Production", animation_frame="Crop_Year",
           animation_group="State_Name", color="State_Name", range_y = [0.75, 5], range_x=[-1E6, 20E6])
```



- It can be inferred that the southern states have better efficiency over the years and from 2008 Madhya Pradesh also shows significant improvement.